Log(lambda) Modifications for Optimal Parallelism

Fabien Teytaud, Olivier Teytaud

Log(λ) Modifications for Optimal Parallelism

F. Teytaud and O. Teytaud

TAO, LRI, UMR 8623(CNRS - Université Paris-Sud),
bat 490 Universite Paris-Sud 91405 Orsay Cedex France, fteytaud@lri.fr

Abstract. It is usually considered that evolutionary algorithms are highly parallel. In fact, the theoretical speed-ups for parallel optimization are far better than empirical results; this suggests that evolutionary algorithms, for large numbers of processors, are not so efficient. In this paper, we show that in many cases automatic parallelization provably provides better results than the standard parallelization consisting of simply increasing the population size λ. A corollary of these results is that logarithmic bounds on the speed-up (as a function of the number of computing units) are tight within constant factors. Importantly, we propose a simple modification, termed log(λ)-correction, which strongly improves several important algorithms when λ is large.

1 Introduction

Evolutionary algorithms (EAs) are well known robust and simple optimization algorithms. It is usually said that EAs are highly parallel, because they are population based [5]. In this paper we study the case for which we have a large number of processors, and we note that the theoretical bounds are far better than the empirical results for the current version of the algorithms in continuous domains. In Section 2 we summarize the state of the art for complexity lower bounds in EA and parallel EAs, especially for the continuous case. Section 3 shows how an optimal speed-up for parallel EAs can be reached; this is an automatic construction of a parallel algorithm with asymptotically optimal speed-up. Section 4 shows that this optimal speed-up is not reached by several well known algorithms. Section 5 shows experimentally the efficiency of parallel algorithms derived from our theoretical analysis; a similar modification, termed log(λ)-correction, is applied to several classical algorithms. Section 6 concludes. Due to length constraints, all proofs have been reported to `http://www.lri.fr/~teytaud/ppsn10long.pdf`.

2 Complexity bounds for evolutionary algorithms

We consider optimization in a domain S, subset of a normed vector space (S might be a non-empty subset of $\mathbb{R}^d$, or bistrings, the theoretical analysis below does not require anything more). For $\epsilon > 0$, we define $N(\epsilon)$ to be the maximum integer n such that there exist n distinct points $x_1, \ldots, x_n \in S$ with $\|x_i - x_j\| \geqslant 2\epsilon$ for all $i \neq j$. In particular, $N(\epsilon) = |S|$ when ϵ is small enough in the case of a

finite domain S, and $\log N(\epsilon) \sim N \log(1/\epsilon)$ when $\epsilon \to 0$ if the domain $S \subset \mathbb{R}^N$ is bounded with non-empty interior. For a domain included in $\mathbb{R}^N$, we then consider the convergence ratio $CR = \frac{\log N(\epsilon)}{N n_{\epsilon,\frac{1}{2}}}$, where:

- $n_{\epsilon,\frac{1}{2}}$ is the number of evaluations necessary for ensuring, with probability at least $\frac{1}{2}$, a distance $||\hat{x} - x^*||$ at most ϵ between the approximation $\hat{x}$ of the optimum and the optimum x^*. The choice of the $\frac{1}{2}$ is arbitrary; other constants lead to similar results.
- N is the dimension of the search space.

A faster algorithm means CR larger. Convergence rate is usually defined as $\exp(-CR)$. Following [13] we prefer the convergence ratio as it is more convenient for expressing speed-ups; the speed-up between two algorithms is just the ratio between their convergence ratios, and the number of iterations for reaching a given precision is proportional to the inverse of the convergence ratio. Table 1 summarizes known bounds on the convergence ratio, and distinguishes:

- (μ, λ)-ES and $(\mu+\lambda)$-ES, respectively non-elitist and elitist Evolution Strategies; in the former case, the μ best points among λ generated points are selected, whereas in the latter case the μ best points among the union of (i) the λ generated points, and (ii) the previous population, are selected.
- full ranking (FR) evolution strategies and selection-based (SB) evolution strategies; in the former case, the optimization algorithm is informed of the complete ranking of the μ selected points, whereas in the latter case the optimization algorithm is only informed of which μ points are the best ones. For example, $(\mu/\mu, \lambda)$-ES, i.e. the new parent is simply the average of the μ best points (intermediate recombination), are selection-based, whereas weighted recombination is full ranking. For $\mu = 1$, there's no difference between FR and SB.

These concepts will be formalized below (Eq. 1-4) and we will study the optimal speed-ups, i.e. the convergence ratio as a function of λ.

3 Automatic speculative parallelization

A solution (in some cases) for automatic parallelization of an algorithm consists in developing the tree of possible futures, to compute separately all branches, and then to discard bad (non chosen) branches. This is a form of speculative parallelization [4]. We here show that this simple approach can be applied to EAs. We have to introduce a somehow tedious formalization; this is necessary for the mathematical formalization of our proofs. As already pointed out in [14], most EAs can be rewritten as follows:

$$(x_{n\lambda+1}^{O_1,O_2}, \ldots, x_{(n+1)\lambda}^{O_1,O_2}) = O_1(\theta, I_n) \quad \text{(generation)} \quad (1)$$

$$\forall i \in [\![n\lambda + 1, (n+1)\lambda]\!], y_i = f(x_i^{O_1,O_2}) \quad \text{(fitness)} \quad (2)$$

$$g_n^{O_1,O_2} = g(y_{n\lambda+1}, \ldots, y_{(n+1)\lambda}) \quad \text{(selection)} \quad (3)$$

$$I_{n+1} = O_2(I_n, \theta, g_n^{O_1,O_2}), \quad \text{(update)} \quad (4)$$

Framework	SB- (μ, λ) -ES	SB- $(\mu + \lambda)$ -ES	FR- (μ, λ) -ES	FR- $(\mu + \lambda)$ -ES
General case	$\frac{1}{N}\left(\lambda - \frac{1}{2}\log(2\pi\lambda)\right)$	$\frac{1}{N}\left(\log\binom{\lambda}{\mu}\right)$	$\frac{1}{N}\left(\lambda - \frac{1}{2}\log(2\pi\lambda)\right)$ $\times \log(\mu!)$	$\frac{1}{N}\left(\log\binom{\lambda}{\mu}\right)$ $\times \log(\mu!)$
VC-dimension V	$\frac{V}{N}\log(\lambda)$	$\frac{V}{N}\log(\lambda + \mu)$	$\frac{V}{N}\left(4\mu + \log(\lambda)\right)$	$\frac{V}{N}\left(4\mu + \log(\lambda)\right)$
Quadratic case	$O\left(N\log(\lambda)\right)$	$O\left(N\log(\lambda + \mu)\right)$	$O\left(N(\mu + \log(\lambda))\right)$	$O\left(N\left(\mu + \log(\lambda)\right)\right)$
Sphere function	$(1 + \frac{1}{N})\log(\lambda)$	$(1 + \frac{1}{N})\log(\mu + \lambda)$	$2\log(\lambda)$	$O(\mu + \log(\lambda))$
Sphere function with $\lambda = 2N$			$\Omega(1)$	$\Omega(1)$

Table 1. Upper bound on the convergence ratio; also some lower bounds on the convergence ratio for $\lambda = 2N$ for the sphere function, in the last row - these lower bounds from [13] show that a linear speed-up can be achieved w.r.t. λ constant for $\lambda = 2N$ (compare with the first row). The first row is the general case [14]; it holds in all cases, and is sometimes better than other rows (when λ is small). The second row is when the level sets of fitness functions have VC-dimension V in $\mathbb{R}^N$. The third row is just the application of the second row to the case of convex quadratic functions ($V = \Theta(N^2)$). The fourth row is the special case of the sphere function [13]. The tightness of the $\log(\lambda)$ dependency will be shown in this paper.

for some fixed O_1, O_2, I_0, some random variable θ, and g with values in a set of cardinality K, where:

- I_0 is the initial state and I_n is the internal state at iteration n;
- θ is the random seed;
- g^{O_1,O_2} is the information used by the algorithm, typically in our case the indices of the selected points (and possibly their ranking in the FR case);
- $x_k^{O_1,O_2}$ is the k^{th} visited point and y_k is its fitness value (y_k should, theoretically, be indexed with O_1, O_2 as well);
- (O_1, O_2) is the optimization algorithm, with:
  - O_1 is the function generating the new population (as a function of the random seed and of the internal state);
  - O_2 is the function updating the internal state as a function of the random seed and of the extracted information g.

(note that $g_n^{O_1,O_2}$ and $x_n^{O_1,O_2}$ both depend on θ and f; we drop the indices for the sake of clarity.) We will term such an optimization algorithm a λ-optimization algorithm; this means that λ fitness values are computed at each iteration. The optimization algorithm is defined by O_1, O_2, I_0, θ; in cases of interest (below) we will use the same θ and the same I_0 for all algorithms and therefore only keep the dependency in O_1 and O_2 in notations. In EAs, g_n has values in a discrete domain; typically, either g_n has values in the set of the finitely many possible ranking of the individuals; or g_n has values in the finite set of possible vectors of ranked indices of selected individuals. g_n is in both cases the only information that the algorithm extracts from the fitness function. In the FR case and $\mu = \lambda$, for example g_n is $(sign(y_{n\lambda+i} - y_{n\lambda+j})_{(i,j)\in[\![1,\lambda]\!]^2})$ where $sign(t) = 1$ for $t \geq 0$ and $sign(t) = -1$ otherwise. In the SB case for (μ, λ)-ES, the formulation is a bit more tedious:

$$g_n = \{I = \{i_1, \ldots, i_\mu\} \subset [\![1, \lambda]\!]^\mu; Card\ I = \mu \text{ and } k \in I \wedge k' \in [\![1, \lambda]\!] \setminus I \Rightarrow y_{n\lambda+k} \leq y_{n\lambda+k'}\}.$$

An important property is that the set of possible values for g_n has cardinality $K < \infty$; K can be bounded as follows:

- (μ, λ)-ES (evolution strategies) with equal weights; then $K \leq \lambda!/(\mu!(\lambda-\mu)!)$;
- (μ, λ)-ES with weights depending on the rank; then $K \leq \lambda!/(\lambda-\mu)!$;
- $(1+\lambda)$-ES; then $K \leq \lambda + 1$;
- $(1, \lambda)$-ES; then $K \leq \lambda$.

K will be termed the *branching factor* of the algorithm. The branching factor, and bounds in Table 1 on the branching factor, have been used in [13] for proving results shown in Table 1; we will use it here for proving lower bounds on the parallelization of EAs; the lower the branching factor, the better the speed-up. We will say that a λ'-optimization algorithm O'_1, O'_2 simulates a λ-optimization algorithm O_1, O_2 with speed-up D if and only if

$$\forall \theta, \forall n \geq 0, \forall i \in [\![1, \lambda]\!], x^{O'_1, O'_2}_{n\lambda'+i} = {x^{O_1, O_2}}_{nD\lambda+i}. \tag{5}$$

θ is the random seed; it is removed of indices for short as discussed above, rigorously all the x's depend on it. We now show how we can automatically build O', which is equivalent to O, but with $\lambda' > \lambda$ evaluations at the same time and a known speed-up.

Theorem 1. (Automatic parallelization of EAs and tightness of the $\log(\lambda)$ speed-up.) *Consider a λ-optimization algorithm (O_1, O_2) as in Eqs 1-4 with branching factor K, and consider λ' such that for some $D \geq 1$:*

$$\lambda \frac{K^D - 1}{K - 1} = \lambda'. \tag{6}$$

Then, there is a λ'-optimization algorithm which simulates (O_1, O_2) with speed-up D.

Remark: The speed-up is therefore $D = \frac{\log(1+\frac{\lambda'}{\lambda}(K-1))}{\log(K)}$.

4 Real world algorithms don't all reach the optimal speed-up

In this section we show that the one-fifth rule, the self-adaptation and the cumulative step-size adaptation all do not reach the optimal speed-up (the optimal speed-up is $\log(\lambda)$ for $\lambda \to \infty$, see Table 1 and [13]) when using the natural parallelization consisting in increasing λ to the number of processors and evaluating one individual per core. More precisely, these classical algorithms have bounded speed-up as a function of λ (i.e. speed-up $O(1)$ as $\lambda \to \infty$). In all sections below, we consider optimization in the continuous domain, with Gaussian mutations and define $\eta^* = \sigma_{n+1}/\sigma_n$ (η^* depends on n, but we will consider a fixed value of n here and therefore we will drop this dependency in the notation η^*).

The main important point is that the convergence rate is lower bounded by η^*; formally, $CR \leq \mathbb{E} - \log(\eta^*)$. Roughly speaking, it is not possible to decrease

the distance to the optimum by z at each iteration (on average, logarithmically), if you don't divide the step-size by z, on average. Therefore, it will be sufficient, in the sequel, to lower-bound η^* for various classical step-size adaptation rules, independently of λ, in order to show that the step-wise adaptation does not provide an optimal convergence rate as $\lambda \to \infty$ (an optimal convergence rate should be $\eta^* = O(1/\log(\lambda))$). More precisely, as η^* is a random variable, we have to show that the expected logarithm of η^*, i.e. $\mathbb{E}\log\eta^* = \mathbb{E}\log(\sigma_{n+1}/\sigma_n)$ is lower bounded by a constant $> -\infty$. The following sections (4.1, 4.2, 4.3) use this fact for showing the poor efficiency of the usual algorithm for $\lambda \to \infty$.

4.1 One-fifth rule

The one-fifth rule [8] is the oldest and most well known algorithm for adapting the step-size. The one-fifth rule can be applied in different manners to $(\mu/\mu, \lambda)$ algorithms. Consider $\hat{p}$ equal to the ratio between (i) the number of generated individuals with fitness better than the center of the Gaussian generating the offspring (ii) the number of generated individuals; $0 \leq \hat{p} \leq 1$. A first possible implementation of the one-fifth rule is

$$\hat{p} \leq 1/5 \Rightarrow \eta^* = K_1 \in]0,1[\text{ and } \hat{p} > 1/5 \Rightarrow \eta^* = K_2 > 1 \quad (7)$$

$$\text{and a second version is } \eta^* = K_3^{(\hat{p}-1/5)} \text{ for some } K_3 > 1. \quad (8)$$

Proposition 1: *The one-fifth rule, implemented as in Eq. 7 or in Eq. 8, has the property that for each iteration n, there is $C > -\infty$ such that $\mathbb{E}\log(\frac{\sigma_{n+1}}{\sigma_n}) > C$.*

Therefore, we have shown that with the one-fifth rule, the convergence ratio (and therefore the convergence rate) is $O(1)$ (as $\lambda \to \infty$; convergence rates and ratios are defined in Section 2).

4.2 Self-adaptation (SA)

The proof of the limited speed-up for SA requires the following lemma.

Lemma: *The expected logarithm of the average (arithmetic or geometric average) of the μ smallest of λ independent standard log-normal random variables, with $\mu/\lambda \to k > 0$ and $\mu > 0$, is lower bounded by some constant $> -\infty$. More formally, if $N_{(1)}, \ldots, N_{(\lambda)}$ are sorted standard independent Gaussian variables, and $L_{(i)}$ is $\exp(N_{(i)})$, then*

$$\inf_{\lambda>0} \mathbb{E}\log\frac{1}{\mu}\sum_{i=1}^{\mu}\exp(N_{(i)}) > -\infty \text{ and } \inf_{\lambda>0} \mathbb{E}\frac{1}{\mu}\sum_{i=1}^{\mu} N_{(i)} > -\infty.$$

Proposition 2. *Consider a SA algorithm in which σ_{n+1} is the average (geometric or arithmetic average) of $\sigma_n \times L_1, \sigma_n \times L_2, \ldots, \sigma_n \times L_\lambda$, for $L_1, \ldots, L_\lambda$ as in the lemma above. Then, there exists some $C > -\infty$ such that $\mathbb{E}\log(\frac{\sigma_{n+1}}{\sigma_n}) > C$.*

Remark: Rescaling the N_i by any constant (equivalently, $L_i = \exp(kN_i)$ for some $k > 0$) does not change the result.

4.3 Cumulative step-size adaptation

It has been experimentally shown in [3] that CMA has a poor speed-up as a function of λ. Empirically, the Estimate of Multivariate Normal Algorithm (EMNA) [7] has a much better behavior, but the speed-up curve becomes constant as a function of λ, instead of logarithmic, for λ large [10]. We here show formally that Cumulative Step-size Adaptation (CSA) does not reach optimal speed-up $\log(\lambda)$. We (classically) formalize an iteration of CSA in dimension N as follows:

$$w_i \geq 0, \sum_{i=1}^{\mu} w_i = 1 \quad (9) \qquad d_\sigma = 1 + 2\max(0, \sqrt{\frac{\mu_{eff}-1}{N+1}} - 1) \quad (12)$$

$$\mu_{eff} = \frac{1}{\sum_{i=1}^{\mu}(w_i^2)} \quad (10) \qquad c_\sigma = \frac{\mu_{eff}+2}{N+\mu_{eff}+3} \quad (13)$$

$$\chi_N > 0 \ , \ ||p_c|| \geq 0 \quad (11) \qquad \sigma_{n+1} = \sigma_n \exp\left(\left(\frac{||p_c||}{\chi_N} - 1\right) \cdot \frac{c_\sigma}{d_\sigma}\right).$$

(||.|| does not have to be a norm, we just need Eq. 11). These assumptions, to the best of our knowledge, hold in all current implementations of CSA. We then show the following

Proposition 3. *For any dimension N, there exists $C > 0$ such that, for any λ, $\eta_n^* = \frac{\sigma_{n+1}}{\sigma_n} \geq C$.*

This proposition shows that $\eta^* \geq \exp(-1)$; this implies that $CR \leq 1$, *i.e.* for cumulative step-size adaptation the speed-up is $O(1)$ for $\lambda \to \infty$.

5 Experimental speed-up

Theorem 1 proves that this automatic parallelization reaches $\log(\lambda)$, which is asymptotically optimal within a constant factor, but there are algorithms for which automatic parallelization works only for λ very large, in particular when the full ranking of selected individuals is used (because in this case the branching factor K is much bigger). Therefore, in this section, we will provide other tricks than the automatic parallelization for ensuring the suitable $\log(\lambda)$ property. In all cases below, we keep a parallelization based on the simple principle of one individual per processor, but we modify either the selection ratio or the step-size adaptation rule, so that this principle leads to much better speed-ups. If the speed-up is bounded, then σ is divided by, at most, a fixed constant, independently of λ. If we want to reach the "$\log(\lambda)$" speed-up, then we must decrease $\log(\lambda)$ by $\Theta(\log(\lambda))$; *i.e.* divide σ by an exponent of λ. We will here apply $\sigma \leftarrow \sigma / \max(1, (\zeta\lambda)^{1/N})$ for some value of ζ. We consider, CMSA, EMNA and CMA-ES. CMA-ES is interesting; as it is a FR-(μ, λ)-ES, and therefore has a big branching factor $K = \lambda!/(\lambda-\mu)!$, and therefore the automatic parallelization becomes efficient only for huge numbers of processors - we will show below simple tricks empirically solving this trouble.

5.1 The $\log(\lambda)$ correction for CMSA

CMSA is the algorithm for which implementing the $\log(\lambda)$ correction is the easiest: we just have to modify the selection ratio μ/λ. We give experimental

results in Fig. 1, and a more detailed presentation and analysis of this correction can be found in [9].

5.2 The log(λ) correction for EMNA

We present results of the isotropic EMNA (the step-size is the same in all directions), on the sphere function. The presented numbers are the mean progress of the log of the distance to the optimum, multiplied by the dimension[1], estimated with the following experimental conditions:

- Column "baseline": the standard EMNA algorithm from [7], with $\mu = \lambda/4$;
- Column "+QR": EMNA, plus the quasi-random mutations as defined in [12];
- Column "+log(λ)": the same as "+QR", except that we add the log(λ) correction, i.e. we modify σ according to formula $\sigma \leftarrow \sigma / \max(1, (0.15\lambda)^{1/N})$ (which ensures that $\log(\sigma)$ decreases by $\tilde{}\log(\lambda)$ as requested above);
- Column "+weighting": the same as "+log(λ)", except that we apply the reweighting as in [11] (this reweighting is based on the density of the Gaussian used for the offspring; variants of reweighting based on the ranks can be found in [1, 2]).

In all cases the initial step-size is $\sigma = 1$ and the initial point is randomly drawn on the unit sphere with radius $\sqrt{N}$ with N the dimension. The 3 following columns provide the p-value of the comparison between a column and the previous column; the significance is very high. Then, the last column presents the normalized convergence rate of the algorithm with QR and reweighting, but without the log(λ)-correction; with this column, we can check that the improvement is due to the log(λ) correction and not to the combination QR+reweighting. This is detailed in Figure 1 (left), with result in Table 2. Interestingly, the log(λ) correction is not efficient if we do not apply the reweighting trick from [11]. This is somewhat natural, as the log(λ) correction strongly increases the risk of premature convergence, which is reduced by the reweighting.

5.3 The log(λ) correction for CMA-ES

We propose to add the following line in CMA, after the computation of σ:

$$\sigma = \sigma / \max(1, (\zeta\lambda)^{1/N}). \tag{14}$$

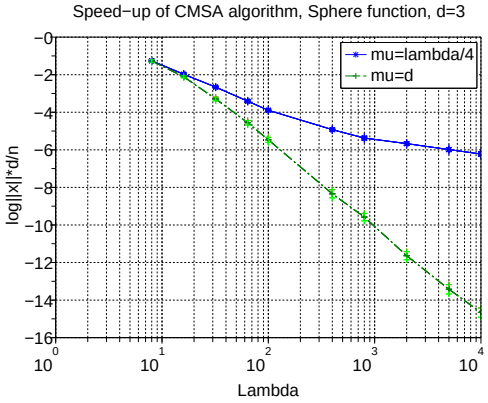
This formula avoids the bad behavior pointed out in Proposition 3 and experimentally strongly improves the results. We consider f as the best fitness found by the algorithm after a fixed number of evaluations. We report the mean of $\frac{N \cdot \log(f)}{\#evaluations}$ and the mean of $\log(f)$ in Fig. 1. The number of function evaluations is $100N^2$. Following [3], we experiment two size of population, $\lambda = 8N$

[1] It is known that the log-distance to the optimum decreases linearly with the dimension; therefore we multiply the results by the dimension in order to have homogeneous results for various dimensions. Following the theoretical analysis in [13], we expect an improvement as the dimension increases, which is confirmed experimentally here.

Dimension, lambda	Baseline	+QR	+log(λ)	+weight	P-value for +QR	P-value for + log(λ)	P-value for +weight	QR+weight but no log(λ)
2,20	-1.61	-1.91	-0.66	-2.43	0.00	1	0	-2.02
2,60	-2.04	-2.13	-0.27	-3.95	0.00	1	0	-2.17
2,200	-2.17	-2.27	-0.17	-5.31	6e-16	1	0	-2.16
2,600	-2.22	-2.27	-0.14	-6.44	4e-15	1	0	-2.27
2,2000	-2.22	-2.38	-0.13	-7.68	0	1	0	-2.32
2,6000	-2.33	-2.51	-0.13	-8.85	0	1	0	-2.38
3,30	-2.09	-2.49	-0.69	-1.67	0.00	1	0	
3,90	-2.43	-2.52	-0.28	-4.58	2e-05	1	0	
3,300	-2.53	-2.59	-0.21	-6.02	0.00	1	0	
3,900	-2.57	-2.71	-0.17	-7.20	5e-09	1	0	
3,3000	-2.65	-2.87	-0.16	-8.52	0	1	0	
3,9000	-2.77	-2.94	-0.15	-9.63	3e-16	1	0	
5,50	-2.72	-2.96	-0.54	-3.28	1e-12	1	0	-2.72
5,150	-3.02	-3.09	-0.42	-5.60	0.00	1	0	-2.85
5,500	-3.08	-3.26	-0.31	-6.97	2e-14	1	0	-3.00
5,1500	-3.22	-3.41	-0.26	-8.19	1e-12	1	0	-3.17
5,5000	-3.35	-3.63	-0.22	-9.56	0	1	0	-3.32
5,15000	-3.53	-3.74	-0.20	-10.84	1e-15	1	0	-3.53
20,200	-5.56	-5.89	-2.52	-2.24	1e-09	1	0.74	-3.30
20,600	-6.05	-6.55	-1.86	-7.57	0	1	0	-4.83
20,2000	-6.81	-7.17	-1.44	-11.27	1e-13	1	0	-6.29
20,6000	-7.25	-7.73	-1.17	-12.98	0	1	0	-6.75
20,20000	-7.71	-8.03	-0.99	-14.62	0	1	0	-7.36
20,60000	-7.93	-8.09	-0.87	-16.17	1e-08	1	0	-7.96
40,400	-8.36	-8.83	-5.35	-1.31	3e-09	1	1	
40,1200	-9.27	-9.54	-4.33	-2.94	8e-05	1	0.97	
40,4000	-10.00	-10.15	-3.47	-8.25	3e-05	1	0	
40,12000	-10.38	-10.48	-2.88	-16.30	0.01	1	0	

Table 2. Convergence rates of EMNA. We see that (i) QR works very well (ii) reweighting does not always improve the results (it has been published as a tool against premature convergence and not as a tool for fastening EMNA) (iii) the log(λ) correction greatly improves the results, but only if reweighting is applied; this is somewhat natural, as, without reweighting, the log(λ) correction increases the risk of premature convergence.

Initialize $\sigma \in \mathbb{R}$, $y \in \mathbb{R}^N$.
while Halting criterion not fulfilled **do**
  for $l = 1..\lambda$ **do**
    $z_l = \sigma N_l(0, Id)$
    $y_l = y + z_l$
    $f_l = f(y_l)$
  end for
  if "Reweighting" version **then**
    Let $w(i) = 1/density(x_i)$
    // with $density$ the proba. density
    // used for generating the offspring.
  else
    Let $w(i) = 1$
  end if
  Sort the indices by increasing fitness:
  $f_{(1)} < f_{(2)} < \cdots < f_{(\lambda)}$.
  $z^{avg} = \frac{1}{\sum_i^\mu w_{(i)}} \sum_{i=1}^{\mu} w_{(i)} z_{(i)}$
  $\sigma = \sqrt{\frac{\sum_{i=1}^{\mu} w(i)||z_{(i)} - z^{avg}||^2}{\sum_{i=1}^{\mu} w(i) \times N}}$
  if $\log(\lambda)$ version **then**
    $\sigma = \sigma / \max(1, (0.15\lambda)^{1/N})$.
  end if
  $y = y + z^{avg}$
end while



λ	CMA	CMA with $\log(\lambda)$-correction
Dimension 2		
$8 \times N$	-0.100±0.001	**-0.177±0.001**
$8 \times N^2$	-0.0741±0.0009	**-0.134±0.001**
Dimension 10		
$8 \times N$	-0.0338±6e-05	**-0.0389±0.0001**
$8 \times N^2$	-0.00971±6e-05	**-0.0174±0.0001**
Dimension 30		
$8 \times N$	-0.0107±1e-05	**-0.0118±2e-05**
$8 \times N^2$	-0.00188±1e-05	**-0.00370±1.e-05**

Fig. 1. Left: The EMNA algorithm with weighted averages. N_l is a Gaussian random variable, or a Gaussian quasi-random variable for "QR" versions (see text). **Right, top:** Example of the limited speed-up of real-world algorithms, and the strong improvement provided by a simple correction. n is the number of iterations; the algorithms run until fitness value 10^{-10} is reached, x is the best point so far. This experiment is done in dimension 3, and we plot the log-distance to the optimum normalized by the dimension and the number of generations of the algorithm (the lower the result, the better). The usual initialization $\frac{\mu}{\lambda} = \frac{1}{4}$ is outperformed, by far, by $\min(d, \lfloor \lambda/4 \rfloor)/\lambda$. **Right, bottom:** Comparison between CMA and CMA with $\log(\lambda)$-correction in various dimensions. The maximum number of function evaluations is 400 (in dimension 2), 10 000 (in dimension 10) and 90 000 (in dimension 30), and the constant ζ involved in the λ correction (Eq. 14) $0.4^{1/2}$ in dimension 2, 1 in dimension 10, $1.3^{1/30}$ in dimension 30. In all cases the λ-correction provides an improvement. Whereas in the case of EMNA we could use the same constant in all cases and the results were very stable as a function of the constant, with CMA we had to modify the constant ζ as a function of the dimension in order to get good results.

and $\lambda = 8N^2$. If the dimension is small (2) we almost have a speed-up of 2 independently of the size of the population. However, if the dimension becomes larger (10 or 30) we have a good speed-up only if the size of the population is large ($\lambda = 8N^2$). The results are good, but not very good, and CMA with this correction is still far from the efficiency of CMSA or EMNA for large population size; we guess however that improvements of our formula above are possible, and also we guess that modifying the rule for computing the new parent should be adapted for λ large.

6 Conclusion

The new results in this paper are as follows. First, we have shown in Section 3 that theoretical bounds in [13] are tight for their dependencies in λ. In particular,

well parameterized algorithms should have a speed-up $\Theta(\log(\lambda))$. Second, we have shown in Section 4 that many current algorithms do not match this tight dependency. Propositions 1, 2 and 3 show that the speed-up is $O(1)$ for the one-fifth rule, the self-adaptation, and cumulative self-adaptation respectively. The tightness is shown by an explicit construction of a parallel version of EA, which can readily be applied also for direct search methods [6] as well; thanks to this explicit construction, we provide an automatic parallelization with, provably, asymptotically better results. Related experimental results are shown in Section 5. They show that parallel algorithms derived from our analysis are faster and in some cases by far than algorithms based on simply increasing λ; moreover the new version is not more difficult to implement.

References

1. D. V. Arnold. Weighted multirecombination evolution strategies. *Theor. Comput. Sci.*, 361(1):18–37, 2006.
2. D. V. Arnold and D. C. Scott Van Wart. Cumulative step length adaptation for evolution strategies using negative recombination weights. In *Proceedings of EvoWorkshops*, pages 545–554, 2008.
3. H.-G. Beyer and B. Sendhoff. Covariance matrix adaptation revisited - the CMSA evolution strategy. In G. Rudolph, T. Jansen, S. M. Lucas, C. Poloni, and N. Beume, editors, *Proceedings of PPSN*, pages 123–132, 2008.
4. B. Calder and G. Reinman. A comparative survey of load speculation architectures. *J. Instruction-Level Parallelism*, 2, 2000.
5. E. Cantú-Paz. *Efficient and accurate parallel genetic algorithms*. Kluwer Academic Publishers, Boston, MA, USA, 2000.
6. A. Conn, K. Scheinberg, and L. Toint. Recent progress in unconstrained nonlinear optimization without derivatives. *Mathematical Programming*, 79:397–414, 1997.
7. P. Larranaga and J. A. Lozano. *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, 2001.
8. I. Rechenberg. *Evolutionstrategie: Optimierung Technischer Systeme nach Prinzipien des Biologischen Evolution*. Fromman-Holzboog Verlag, Stuttgart, 1973.
9. F. Teytaud. How we can derive parameters of ES for parallel optimization. In *EvoStar 2010*, Accepted.
10. F. Teytaud and O. Teytaud. On the parallel speed-up of Estimation of Multivariate Normal Algorithm and Evolution Strategies. In *Proceedings of EvoStar 2009*, pages 655–664, 2009.
11. F. Teytaud and O. Teytaud. Why one must use reweighting in estimation of distribution algorithms. In *Proceedings of Gecco*, pages 453–460, 2009.
12. O. Teytaud. When does quasi-random work?. In G. Rudolph, T. Jansen, S. M. Lucas, C. Poloni, and N. Beume, editors, *PPSN*, volume 5199 of *Lecture Notes in Computer Science*, pages 325–336. Springer, 2008.
13. O. Teytaud and H. Fournier. Lower bounds for evolution strategies using VC-dimension. In *Proceedings of PPSN*, pages 102–111, 2008.
14. O. Teytaud and S. Gelly. General lower bounds for evolutionary computation. In *proceedings of PPSN*, pages 21–31, 2006.